Machine Translation Detection - GitHub

Student: Sabhya Chhabria

Date: February 5, 2021

Data Pipeline + Feature Engineering

In order to load the data, I created a custom **PyTorch Dataset** and then used **PyTorch Dataloaders** to serve the train and test data. The raw text is passed through a feature engineering process. The features themselves consist of various metrics that evaluate the quality of machine-translated text. I will now justify why I made use of each features. In total, I used 9 features, 7 of which were variations of the BLEU score and the other two are GLEU scores and NIST scores. For the BLEU scores, I made use of both individual and cumulative scores. This included unigram, bigram, trigram, and quadrigram scores for both score types. Note that unigram scores are the same for individual and cumulative types (Nguyen-Son et al.). I also include the GLEU score in the feature set since the BLEU score was originally designed as a corpus measure rather than a sentence level measure, which gives it some undesirable properties (Wu et al.). I further included the NIST score since it weights the rarity of the n-gram when calculating the score (Doddington).

Models

I decided to use two models. A Support Vector Classifier (SVC) and a Feed Forward Neural Network (FFNN). I decided to use the SVC (from **sklearn**) because it was used in the (Nguyen-Son et al.) paper, which made use of similar features. I then implemented an FFNN in PyTorch. While training, I made use of a hyperparameter optimization PyTorch package called Optuna, which helped me maximize the $F1$ scores through the ideal parameters.

Results

SVC: The SVC resulted a maximum $F1$ score of 0.750. The parameters that I experimented with were the kernel and value of C. The best SVC configuration used $C = 1$ and kernel = rbf (radial basis function). The results are on par with the results that were seen in the (Nguyen-Son et al.) paper.

FFNN: The FFNN resulted in a maximum $F1$ score of 0.756, which was marginally better than the SVC. The parameters that I experimented with are the hidden dimension of the network, activation function, dropout amount, and optimizer and its learning rate. The configuration of the best performing model are hidden dimension = 128, activation = sigmoid, dropout = 0.1, optimizer = Adam, learning rate = 0.01.

I believe thet $FFNN$ performed slightly better since it has more parameters, makes use of a better optimzier, and uses droput to tune network performance.

$F1$ values that were yielded for different parameters for the SVC model.

number	svc_c	svc_kernel	f1 score
0	1.0	rbf	0.7500000000000001
1	0.1	sigmoid	0.15384615384615385
2	1.0	poly	0.7291666666666666
3	0.1	sigmoid	0.15384615384615385
4	0.1	rbf	0.7303370786516854
5	0.1	sigmoid	0.15384615384615385
6	0.1	rbf	0.7303370786516854
7	1.0	sigmoid	0.35443037974683544
8	0.1	poly	0.7461139896373058
9	10.0	sigmoid	0.3592814371257485

$F1$ values that were yielded for different parameters for the FFNN model.

number	activation	dropout	hidden_dim	optimizer	learning rate	f1 score
0	relu	0.05	128	Adam	0.1	0.7011494252873564
1	sigmoid	0.1	64	Adam	0.1	0.718232044198895
2	relu	0.05	64	AdamW	0.01	0.7362637362637363
3	relu	0.05	128	Adam	0.01	0.708994708994709
4	sigmoid	0.1	128	Adam	0.01	0.7222222222222222
5	sigmoid	0.1	128	Adam	0.01	0.7422680412371134
6	sigmoid	0.05	64	Adam	0.01	0.7431693989071039
7	sigmoid	0.05	64	AdamW	0.1	0.7282051282051283
8	relu	0.1	64	AdamW	0.1	0.7039106145251396
9	sigmoid	0.1	128	AdamW	0.1	0.6878980891719746
10	tanh	0.05	64	Adam	0.01	0.7199999999999999
11	sigmoid	0.1	128	Adam	0.01	0.7555555555555555
12	tanh	0.1	64	Adam	0.01	0.7113402061855669
13	sigmoid	0.05	128	Adam	0.01	0.7294117647058823
14	sigmoid	0.05	64	Adam	0.01	0.7349397590361447
15	sigmoid	0.1	128	Adam	0.01	0.7455621301775147